CHAPTER 3

Memory and I/O Power Management

CPUs cannot operate effectively without memory to store working data and I/O interfaces to bring in data from drives and the network. Although the CPU is a common focus for power management and power efficiency discussions, in many systems the memory subsystem can contribute a significant power footprint. I/O is also important but tends to contribute a much smaller piece of the pie. This chapter will provide an overview of server memory architecture and how the power and thermal management techniques work. It will also discuss how power is managed for the other I/Os that provide the CPU with the data required for operation.

System Memory

Memory power can contribute a very large +-percentage of the overall platform power in some system designs. Different usage models require wide ranges of memory capacity, causing the importance of memory power to vary from user to user. Different types of memory can also have a wide range of power consumption. This section will provide an overview of memory architecture and how it impacts power consumption in the data center.

Memory Architecture Basics

Before we discuss the power management capabilities of server systems, it is important to understand the basics of how memory works and how power is consumed. Let's start at the high level. Sticks of memory, or DIMMs, are plugged into slots on the platform. Each slot is connected to a *memory channel*. Multiple DIMMs can be connected to the same memory channel, but when this is done, those DIMMs share the same command/data connection, and therefore allow for increased capacity, but not bandwidth. The number of DIMMs per channel is abbreviated as DPC (e.g., 1 DPC = 1 DIMM per channel).

When the CPU issues a read to memory, it generally[1] fetches 64 Bytes (B)[2] of data from a single stick of memory. Each physical address (PA) in the system is mapped to a specific channel/DIMM that is connected into the CPU. The read/write is issued on the command bus, and data is returned (if a read) or sent along with the write command on the data bus. The relevant DIMM on the channel determines that the command is for it and processes the request.

Data on DDR3/DDR4 is handled in Burst-Length 8 (BL8). This means that a single access (read or write) uses eight slots on the memory bus (see Figure 3-1). The memory bus is 8 B wide, providing 64 B of data across these eight *bursts* and runs at the DDR frequency. So, a given channel can provide *DDR Frequency* (*GHz*) * 8 (*Bytes/clock*) of memory bandwidth (in GB/s). Each of these eight bursts will acquire some data from multiple *devices* on the DIMM (the exact number depends on the type of DIMM).

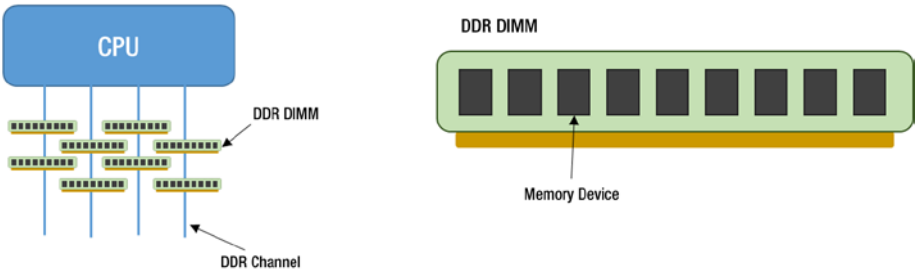

Figure 3-1. *DDR and the CPU platform*

Devices and Ranks

Figure 3-2 provides a high-level overview of memory DIMMs and how they connect to a CPU Socket. Each DIMM of memory consists of a number of memory devices. The devices are the actual "chips" that you will see soldered down to the DIMM. Each device supplies a subset of the 8 B chunks of data that are returned in each burst. Server memory devices/DIMMs can be ×4 or ×8 (called "by 4" or "by 8").[3] This refers to the amount of data that each device supplies toward each 8 B burst. ×4 memory supplies only 4 b of data for each 8 B chunk, and therefore 16 devices are required in order to supply the data. ×8 memory supplies 8 b of data, so only 8 devices are required. Device manufacturers commonly only produce a couple of device sizes at a time—and those devices can either be manufactured into ×8 or ×4 memory. ×4 devices allow for higher DIMM capacities using the same device size as well as improved reliability with error correcting code (ECC) by requiring more devices to supply data for a single 64 B access.

[1]Certain reliability features do exist, like memory Lockstep, which allow for a given 64 B chunk of data to be fetched from multiple devices in order to improve reliability. These are not commonly used in typical servers and are targeted at very high-availability systems.

[2]A bit (b) of data refers to a single binary piece of data (1 or 0). A byte (B) of data refers to a collection of 8 bits.

[3]Other types exist, but are not common in server usage models (e.g., client devices using DDR3 commonly supported ×16 memory as well).

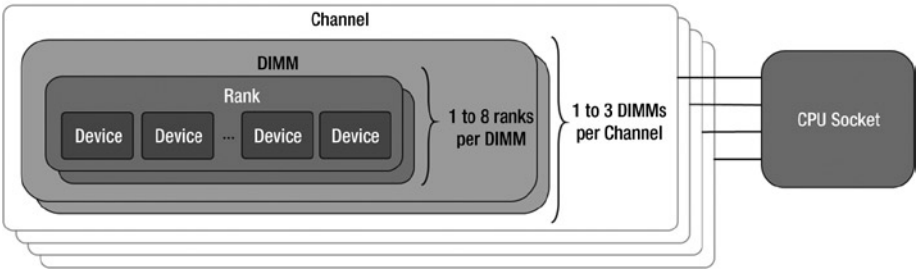

Figure 3-2. *DDR channels, DIMMs, and ranks*

DIMMs also have *ranks* (usually one, two, four, or eight). Individual DRAM devices are single-ranked. So, when a DIMM supports two ranks, for example, this means that the number of DRAM devices on that DIMM is doubled. So, a ×8 DIMM that requires 8 devices to supply 64 B of data will actually have 16 devices if it has two ranks, or 32 devices if it has four ranks. A single bus connects the DIMM to the CPU, and all the ranks on that DIMM share that bus. However, each rank is able to operate somewhat autonomously from the others. One of the biggest challenges for a DRAM device is switching between doing reads and writes (and back again). As a result, the memory controller must insert a sizeable bubble between these types of transactions to a given rank. DIMMs that support multiple ranks are able to sneak other transaction onto the channel bus while one rank switches modes, allowing for improved bandwidth and average latency. As a result, for performance, we generally recommend that you use either two single-ranked (SR) DIMMs of the same size on a channel, or one or more dual-ranked (DR) DIMMs in order to help hide these inefficiencies. Quad-ranked (QR) and even some oct-rank (OR) DIMMs also exist on the market.

■ **Note** All memory is not equal. As with CPUs, new process technologies are being applied to memory, which enables lower power and increased capacities. At a given point in time, vendors are only able to economically manufacture devices up to a given size. The "next size up" (two times the capacity) are at times available, but they come at a significant cost premium. By increasing the number of ranks or moving from ×8 to ×4 devices, vendors can increase the number of devices on a DIMM and thus increase capacity. Using a smaller number of higher capacity devices can consume significantly less power than using a large number of lower capacity devices, despite the fact that both can provide the same memory capacity.

Memory Error Correction (ECC)

Server memory typically leverages ECC memory to provide protection, both from transient errors and device failures. Each 8 B burst of data that is supplied includes an additional ninth byte of data that provides the ECC protection. This increases the cost and power of memory proportionally but is generally considered a "must-have" in server deployments.

■ **Note** It is not possible to disable the ECC device in order to save power in systems today.

With ×8 memory, an additional device is included on the DIMM to support ECC. With ×4, two additional devices are required. ×4 memory provides improved reliability because each device provides a smaller percentage of the data. If a single device fails, the ECC algorithm is able to correct all data and continue to operate the system. On ×8 memory, if a device fails, it is possible to detect such a condition and hang the system, but correction is not possible.

Memory Capacity

The capacity of a DIMM is a function of the ranks, devices, and device size. DIMMs are typically sold in Gigabytes (GB), whereas devices are typically referred to in Gigabits (Gb). The following formula summarizes how one can calculate the capacity of a DIMM based on the components:

$$Capacity\,(GB) = \frac{\left(DeviceSize\,(Gb) * Ranks * \frac{Devices}{Rank}\right)}{\left(\frac{8\ bits}{Byte}\right)}$$

$$= \frac{\left(DeviceSize\,(Gb) * Ranks * \frac{64B}{byX}\right)}{\left(\frac{8\ bits}{Byte}\right)}$$

$$= \frac{\left(DeviceSize\,(Gb) * Ranks * 8\right)}{byX}$$

Examples:

- DR ×8 4 Gb = 4 Gb * 2 ranks * 8 / (×8) = 8 GB
- QR ×4 8 Gb = 8 Gb * 4 ranks * 8 / (×4) = 64 GB

One can increase DIMM capacity by increasing the number of ranks, the device size, or the number of devices per DIMM (moving from ×8 to ×4).

■ **Note** For optimal performance, we typically recommend that you use dual-ranked (DR) memory, because it allows for more efficient use of the memory bus. This is particularly true of workloads that make use of high memory bandwidth. Single-ranked memory can also show good performance when you use it with multiple DIMMs of the same size per channel (since the memory controller has more than one rank to work with). Quad-ranked (QR) memory also can make efficient use of the memory bus, but it typically requires a lower frequency.

Device Power Characteristics

Many users have a feel for how much memory they think they need, but it can be difficult to understand how to populate the system in order to provide that desired memory. Table 3-1 provides some general rules of thumb for how memory power scales. In each of these cases, capacity is increased by either 1.5 or 2 times. You might expect power to scale directly with capacity, but different decisions result in different power impacts. (Note that these numbers are intended only as a conceptual guidance, not as a hard rule.)

Table 3-1. *DDR4 DIMM Power Scaling Examples*

Parameter	Power Impact	Capacity	Other Notes
Single-rank to dual-rank	~1.3–1.5 times	2 times	Improved performance with a one-DPC configuration, particularly with high-bandwidth workloads.
×8 to ×4	~1.4 times	2 times	Improved reliability.
Two times device capacity	<1.1 times	2 times	Device size increases commonly come with better process technology, so this is difficult to accurately quantify.
One DPC to two DPC	~1.5–1.7 times	2 times	Improved performance with single-ranked DIMMs. Can decrease memory frequency. Power does not double in this case, because the bandwidth provided by each DIMM is 50% of the bandwidth if two DIMMs were used.
Two DPC to three DPC	See note	1.5 times	Generally has significant impact on memory frequency. Not a fair comparison. Three DPC should be used for customers who need capacities not economically possible with two DPC.

■ **Note** Using dual-ranked memory in a one-DPC configuration is typically the most power/performance efficient across a range of workloads. Increasing the number of DIMMs per channel tends to be less power efficient than other alternatives but can also be attractive from a DIMM cost perspective. If low capacity and low bandwidth are required, one-DPC single-ranked topologies are the most power efficient. However, this efficiency quickly falls off if memory bandwidth begins to get stressed. Before purchasing one-DPC single-ranked systems, we highly recommend that you characterize the bandwidth requirements of their workloads first (as described in Chapter 7).

DDR has been optimized to minimize leakage power. Not only does this result in minimal power scaling with temperature, but it also minimizes the power cost of increasing the device capacity. This tends to be the most power-efficient mechanism for increasing capacity but can also be price prohibitive, especially after a certain point.

Power deltas for additional ranks and DIMMs tend to be smaller at higher bandwidths since the overheads are amortized across the power for providing the necessary bandwidth. This is not the case with ×8 to ×4 scaling.

DDR3 vs. DDR4

At a high-level, the architecture of DDR3 and DDR4 are very similar. From an end-user perspective, DDR4 enables higher frequencies while running at lower voltages and consuming less power. There are some other internal changes for improving performance (e.g., more banks). Table 3-2 shows how memory technology has progressed in recent years. Voltage has decreased despite increases in maximum frequency. Larger and larger devices have also been possible as process technology shrinks. Note that the maximum device capacities do not always represent what a given processor can support.

Table 3-2. *DDR Generation Comparisons*

DDR Generation	Voltage	Frequencies	Device Capacities
DDR2	1.8 V	up to ~1600	up to 1 Gb
DDR3	1.5 V	up to ~1866/2133	Spec supports up to 8 Gb (4 Gb common)
DDR3L	1.35 V	up to ~1333/1600	
DDR4	1.2 V	up to ~3200 (TBD)	Spec supports up to 16 Gb

DDR3 supported two different voltages: 1.35 V and 1.5 V. However, running at the lower frequency reduced the peak frequency that could be achieved. DDR4 transitioned all memory to use 1.2 V but also provided a significant boost in peak frequencies (and with it, peak bandwidth).

DDR voltage plays a significant role in energy efficiency (the exact amount varies a good amount across memory types). On processors supporting DDR3/DDR3L memory, a tradeoff could generally be made between selecting DDR3 memory and achieving a higher frequency, or DDR3L and operating at a lower voltage. Higher frequencies can significantly improve memory bandwidth and performance on certain workloads (particularly in the high performance computing space). They also provide slightly lower latencies, but these benefits tend to be small (a few percent of performance at best). Many enterprise systems with large memory capacities come nowhere close to the memory bandwidth limits of the system and can save significant energy by using DDR3L with minimal impact to performance. In platforms with smaller memory topologies, the importance of memory voltage overall is much smaller.

RDIMMs, UDIMMs, SODIMMs, and LRDIMMs

Most servers typically use *registered DIMMs* (RDIMMs), although *unregistered DIMMS* (UDIMMs) are an alternative. Signal integrity is challenging with high-speed memory, particularly when multiple memory DIMMs coexist on the same memory channel. RDIMMs include a register on the DIMM that reduces the electrical load on the memory controller; this improves signal integrity and allows for increased memory frequencies (and higher performance). RDIMMs have traditionally been more expensive than UDIMMs because of the smaller volume and additional components. However, this trend may or may not continue as more client devices move to different memory technologies than those found on servers. The register also consumes measurable power (on the order of ~0.5 to ~1 W per DIMM) when active (some of this power can be saved in low-power states). Different processors and platforms have varied rules and constraints about the maximum frequency that can be supported by different types of memory (UDIMM vs. RDIMM) as well as the topology of memory (number of ranks, number of DIMMs per channel, etc.).

UDIMMs can be purchased with and without ECC. ECC increases the number of devices required by 12.5%, and power increases at about the same rate. Small-outline DIMMs (SODIMMs) show very similar characteristics to UDIMMs (both ECC and non-ECC)—they are just physically smaller and therefore cannot hold as many devices.

ECC UDIMMs and SODIMMs are a good solution for low capacity, low power, and low cost deployments. Systems requiring larger capacities or high reliability typically use RDIMMs.

LRDIMMs (load-reduced DIMMs), a type of RDIMM, provide some additional buffering that allows them to provide access to a large number of devices and still maintain high frequencies. QR RDIMMs suffer from electrical issues that limit their frequencies. LRDIMMs attempt to address this by providing an additional buffer chip on the DIMM to address the increased device count and improve signal integrity. LRDIMMs provide high capacity and high performance. The additional buffer chip consumes some additional power and increases memory latencies slightly, but neither is really of consequence at the platform level. LRDIMMs are generally more expensive per GB of memory compared to normal DR RDIMMs. When they were originally released, they also had a healthy price premium over high capacity QR RDIMMs, but that price delta has come down over time (on DDR3). If you need massive memory capacity, LRDIMM is a good place to start.

Memory Channel Interleave and Imbalanced Memory Configurations

Each channel has a finite amount of memory bandwidth that it can sustain. By interleaving a stream of requests across multiple channels, high memory bandwidth (and higher performance) can be achieved. In order to get optimal performance in a system, each channel should be populated with the same capacity of memory. However, sometimes this is not the most cost effective way to achieve a given desired memory capacity. If imbalanced configurations are going to be used, it is best to avoid situations where a single channel has more capacity than the others, because this results in a section of memory with a "one-way" interleave (and 25% of the theoretical peak bandwidth) as shown in Figure 3-3.

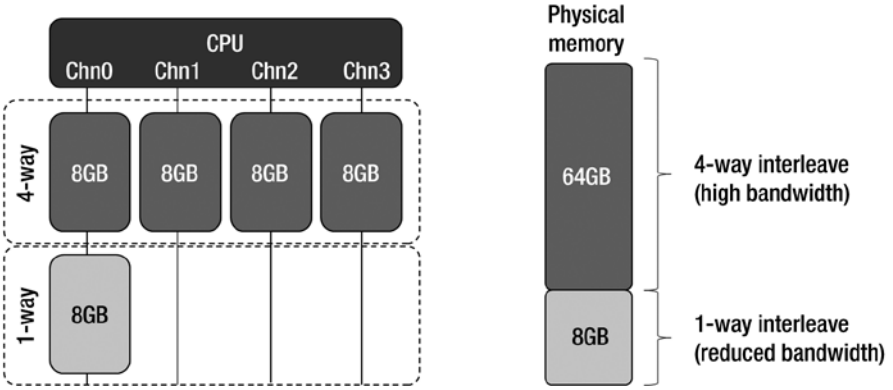

Figure 3-3. *Imbalanced memory interleave example*

By default, most systems today are set up to have separate non-uniform memory access (NUMA) memory regions assigned to each socket. For example, a system with 32 GB of memory would have the first 16 GB of memory allocated on socket 0 and the second 16 GB on socket 1. An alternative to this approach is to use a uniform memory access (UMA) allocation, which interleaves every other cache line across the sockets (effectively providing a single 32 GB region across both sockets in the previous example). In many usage models, this is detrimental to performance because it increases the latency of half of the requests by forcing them to the remote socket. Many users are better off letting the OS (which is aware of this behavior) manage memory allocation and attempt to locate memory on the local socket in order to reduce memory latency. However, certain usage models do exist where the memory NUMA schemes are unable to successfully locate memory in the optimal place and actually lose performance by trying to do so. This is, however, generally uncommon, and most workloads either benefit from such an allocation or show little sensitivity.

Memory interleave does not typically have a direct impact on memory power savings. Power efficiency here is typically achieved by optimizing for performance. Some interesting effects are possible, although in imbalanced configurations. For example, imbalanced configurations can result in certain ranks being accessed infrequently, resulting in higher CKE power savings (discussed more in the following pages).

Power and Performance States

A number of power savings techniques[4] exist for reducing memory power when it is not fully utilized. In general, most of these techniques fit into the category of "turning things off" and not "turning them down." There are really two main techniques for saving power:

- Turning off CKE (clock enable): Power savings during short idle periods at the rank granularity
- Self-refresh: Power savings during long idle periods at the channel granularity

Self-refresh allows for significant memory power to be saved but also can require non-trivial wakeup costs (~10 μs). Turning off CKE provides less power savings but can have very fast wakeups (~10 ns). Turning off CKE can also be done on a rank-by-rank basis, whereas self-refresh must be performed at the channel granularity. As a result, in servers, self-refresh is typically targeted at idle systems, whereas CKE is targeted at moderately active systems. Dynamically managing memory frequency at runtime has not been productized. Changing frequency is a non-trivial piece of work, and the power savings are generally not significant due to the static voltage.

CKE Power Savings

Each rank has a clock enable (CKE) signal that is driven from the CPU memory controller to the DIMM. By de-asserting CKE, the rank is allowed to enter a low power state that can be exited quickly (~10 ns to ~100 ns). A number of different flavors of CKE have differences in their details, but in general, they mostly behave the same. Because CKE is managed on a per-rank granularity, there is potentially more opportunity for CKE power savings on systems with more ranks.

At a high-level, there are two types of CKE:

- Active power down (APD): Memory pages are kept open[5] and the row buffer stays powered up.
- Precharge power down (PPD): The memory pages in all banks on a rank have been "closed" or "precharged," and the row buffer can be powered down.

At first glance, this may sound like a simple power/performance tradeoff. APD saves a bit less power but keeps pages open. However, in practice on servers, it does not actually work out this way. Many times when a rank goes idle for long enough to turn off CKE, the memory pages are also finished being accessed, and therefore having them

[4]See the DDR3 and DDR4 specifications at www.jedec.org for more details. JESD79-4A contains information for DDR4, and JESD79-3F for DDR3.

[5]Memory pages are different from software/TLB pages. Different device types have different memory page sizes. A single 4 K software page can be mapped to either a single memory page (in open page configurations) or to many memory pages (in closed page topologies). Large pages (2 M and larger) typically exist over multiple memory pages.

closed is good for both performance and power. Both CKE PPD and CKE APD are able to save on the order of 30% of the power, and the differences between the two for both power and performance are negligible.

The CPU can force PPD to be used by issuing a PREALL command to a given rank before de-asserting CKE. This closes all the pages in all banks on the rank, allowing PPD to be used. Alternatively, the CPU can simply de-assert CKE when all necessary timing parameters have been met. If one or more pages are open, the DIMM will be in APD. Otherwise, it will be in PPD. In Intel server documentation, the *PPD mode* refers to the case where PREALL is explicitly issued before de-asserting CKE, while *APD mode* disables this PREALL. It is still possible to get into a PPD state from the APD mode if all pages happened to be closed at the time CKE was de-asserted.

On DDR3, there were two main versions of PPD: PPDF (fast) and PPDS (slow). PPDS saved more power at the cost of a slight increase in exit latency. Usually the added exit latency is trivial, so PPDS is generally the better state. In the big picture, the differences between PPDF, PPDS, and APD are not large (either for performance or power efficiency).

One of the big changes with the transition from DDR3 and DDR4 is with how ODT is handled. Rather than requiring the memory controller to manage ODT, it is handled autonomously by the DIMM. In addition to this, the PPDS and PPDF states have been merged into a single PPD state where the DLL is kept powered and ODT is managed by the DIMM. This new simplified mode has excellent power savings. The DLL was redesigned on DDR4 and consumes much less power when active. It is possible to turn off all the DLLs on a channel when an entire channel is idle and save additional power. However, this state has not been productized on Intel servers thus far.

When all ranks on a DIMM are powered down, the register on RDIMMs can also enter a low power state. This does not save all of the register power but can save a couple hundred mW per DIMM.

It is also possible to power down the IBT (input-buffer termination). IBT OFF theoretically can save ~100 mW per DIMM with CKE (it also exists with self-refresh). However, in practice, the savings tend to be much smaller at the platform level because of the increased exit latency. This mode has not been aggressively enabled on servers due to the low power savings upside and wake latency exposures. It is more interesting in microserver usage models, particularly with self-refresh (more later).

Table 3-3 provides a summary of the various types of CKE power savings, including on which types of DDR they are supported. Table 3-4 provides a summary of how CKE has evolved over multiple processor generations.

Table 3-3. *CKE Mode Summary*

Type	Granularity	Banks	ODT	DLL	DDR3	DDR4
APD	Per rank	≥ 1 active	On	On	Supported	Supported
PPDF	Per rank	All precharged	On	On	Supported	Supported (PPD)
PPDS	Per rank	All precharged	Off	Off	Supported	Not Supported

Table 3-4. *CKE Across Generations*

Generation	DDR	APD	PPDF	PPDS	Channel PPDS-DLLOFF
Nehalem/Westmere E5	DDR3	Y	Y	1 DPC	N
Sandy Bridge/Ivy Bridge E5	DDR3	Y	Y	Y	Y
Avoton	DDR3	Y	Y	1 DPC	N
Haswell E5	DDR4	Y	N	N	N

Self-Refresh

DRAM devices (unlike SRAM) must be periodically *refreshed* in order to keep the data valid. Refreshing memory is really nothing more than reading it out of the arrays and writing it back in. During normal operation, the memory controller is periodically issuing refresh commands in order to refresh a portion of the device. The entire device must be refreshed periodically (usually on the order of 10s of milliseconds). When a given channel is not being used, it is possible to put all the DIMMs on that channel into a *self-refresh* state where the DIMM itself is responsible for handling refresh. This state both saves power on the DIMM and allows for additional power to be saved in the CPU memory controller and I/Os. However, this additional power savings generally comes with a non-trivial latency cost. Like with CKE, there are different flavors of self-refresh that provide varied power savings and latency characteristics.

Because self-refresh is performed at the channel granularity and because it tends to have longer exit latencies, it is typically used for saving power when the system is completely idle. Self-refresh residencies in active systems tend to be very low. Typical high-capacity server DIMMs that are in self-refresh tend to consume on the order of 0.2 W to 0.5 W.

The main differentiator between the different flavors of self-refresh is how the CK signals are handled. This is referred to as the *clock stop mode*. CK and CK# are a pair of differential clocks that are necessary for transmitting commands and data between the CPU and memory. If the CPU continues to drive these signals during self-refresh, the wakeup latency can be relatively fast (< 1 μs). However, this mode saves minimal additional power compared to simply turning off CKE and the DLL. The clock can also be stopped. It can be tri-stated, driven low, or driven high. Each of these states results in additional power savings, but exit latency increases to ~10 μs.

CKE can also be tri-stated (i.e., not driven to either 0 or 1) during self-refresh to save some additional power (compared to driving it low). It must be driven low on UDIMMs, but otherwise it can be tri-stated (the voltage is not driven high or low; it is simply left to float to wherever it settles).

■ **Note** Self-refresh is most useful when the entire system is idle and CK can be stopped. As a result, it is used sparingly when cores/IO are active (where CKE management is used instead). However, when the entire system is idle, it can be used more aggressively. This is particularly the case when it is paired with package C-state power savings features. In these cases, the 10 µs wake latency can frequently be done in parallel with other long-latency operations (ramping voltage, locking PLLs, etc.), making the power savings effectively "free."

Voltage/Frequency

Systems today will not dynamically change the voltage/frequency of DDR. On DDR3, some devices support running at both 1.5 V and 1.35 V (called DDR3L). With the transition to DDR4, all DIMMs run at 1.2 V. DDR3L was released after DDR3, and DDR4L is expected in the future as well.

Running DDR3 at 1.35 V generally exhibits significant memory power savings. The amount/percentage is very sensitive to the configuration in question, but using DDR3L can save significant power on systems that leverage a large amount of memory.

On the other hand, generally the frequency of memory is really not all that important. Running at lower voltages can limit the achievable frequency in the system on DDR3, and the frequency can impact the maximum amount of power that a DIMM can consume, but the power to run most workloads is typically not that sensitive to frequency. As an example, taking some DDR3L memory that typically runs at 1333 and decreasing the frequency to 1066 and running at the same (moderate) throughput would save less than 5% memory power. At the same time, such a change could also reduce memory CKE residency or increase core active time, further reducing the power benefits from the reduced frequency. With that said, running at 1333 does provide an additional 25% bandwidth, and if that bandwidth is actually used, then the memory power will increase by ~10%–20%. However, this is generally a great power/performance tradeoff—25% more used bandwidth usually means 25% more performance. The 10%–20% memory power increase for 25% more performance is a small power price to pay when measured at the wall.

On DDR4, the percent power savings by reducing frequency is larger, but this is largely because the overall power has gone down. Power savings with DDR4 is typically on the order of 50–400 mW per DIMM when reducing by a single frequency bin (again, without taking into account additional power consumed elsewhere as a result of the lower performance). Long story short: reducing memory frequency is generally not a good idea.

DDR Thermal Management

Managing the temperature of memory DIMMs is critical to preventing loss of data or system crashes. Most server memory is capable of monitoring temperature, but the CPU is responsible for providing the thermal management algorithms that protect the DIMMs. Memory temperature is another input to the fan speed control algorithms that are discussed in Chapter 4.

Monitoring Temperature

The DIMMs themselves typically provide a thermal sensor called a thermal sensor on-die (TSOD), which provides a single temperature reading for an entire stick of memory. Historically, not all memory used in servers included a TSOD (UDIMMs in particular), but as time has progressed it has become standard. There is only a single thermal sensor on a DIMM, and it is commonly located near the center of the DIMM (lengthwise). Air commonly flows down the DIMM and heats up as it passes over the devices. As a result, the first device tends to be at a lower temperature than the last device, with a single temperature reading taken somewhere in the middle. The CPU (or BMC) reads this temperature over a System Management Bus (SMBus).

Memory Throttling

The CPU is responsible for throttling requests to the DIMM in order to reduce the memory temperature when it begins to enter a high temperature range. Table 3-5 provides a summary of some of the common methodologies for management memory thermals.

Table 3-5. *Memory Thermal Management Techniques*

Mechanism	Requires TSOD	Description
OLTT	No	**Open-Loop Thermal Throttling** OLTT is the simplest mechanism for managing thermals. A static bandwidth limit is put in place in an attempt to avoid high-power operation.
CLTT	Yes	**Closed-Loop Thermal Throttling** CLTT takes temperature readings from the TSOD and performs varied levels of memory bandwidth throttling in order to keep the DIMM in a safe operating range.
Dynamic CLTT	Yes	**Dynamic CLTT** Dynamic CLTT is an enhanced version of CLTT that takes other platform information into account (such as fan speed) to adjust the throttling configuration dynamically to save additional power.

OLTT (open-loop thermal throttling) is the most basic mechanism for performing throttling. Historically it was used in low-cost systems that did not have TSODs available on the DIMMs to provide temperature-based throttling. TSODs are standard on most server memory today, but the legacy OLTT mechanisms are still available.

CLTT (closed-loop thermal throttling) is the standard mechanism for providing memory temperature protection. The CPU monitors the temperature of the DIMMs and engages varied levels of throttling depending on the temperature. Doubling the memory refresh rate is also commonly performed at higher temperatures in order to avoid data corruption.

As discussed previously, temperature increases as air flows down the length of a DIMM. This increase is called a *thermal gradient*. The amount of gradient can vary with other platform parameters. For example, high fan speed results in more air flow and smaller temperature gradients than reduced fan speeds. With the baseline CLTT support, platform designers must assume some amount of gradient when configuring the CLTT throttling algorithms. The CPU provides an interface with Dynamic CLTT for the platform management firmware to dynamically change the throttling constraints based on an estimate of the thermal gradient. This can be used to save power or to prevent throttling when fan speeds are high and there is a smaller gradient. The algorithms used to estimate the gradient and configure the throttler are typically proprietary IP for a platform designer.

MEMHOT is a platform signal similar to PROCHOT. Different products make different use of MEMHOT. It can be used as an input to the CPU, providing an indication from the platform that the CPU should perform memory throttling. This input can be used for the platform to trigger memory throttling when a thermal issue is detected in the platform (even if the DIMMs themselves are not too hot). It is also frequently used to throttle memory power/thermals when some other undesirable event is detected in the platform like an overheating power supply. MEMHOT can be used as an output from the CPU and as an indication to the platform management that the DIMMs have reached a high temperature. On some CPUs, MEMHOT can also be bidirectional and support both input and output modes simultaneously.

DDR3 and DDR4 memory also supports an EVENT# pin that triggers when the TSOD detects high temperatures. This open-drain pin is typically wired directly to the BMC and is not used directly by the CPU. It is commonly used for detecting critical temperature levels that require an immediate system shutdown.

CPU DDRIO

I/Os exist on the CPU that connect to the traces that go to the DIMMs. These I/Os typically run at the same voltage as the memory and are supplied by the same voltage regulator.[6] Multiple channels frequently share a voltage regulator. DDRIO power at a first order is a function of the bandwidth that it is driving (both reads and writes). There is some additional power cost that results from increasing the number of DIMMs, but this is not a first-order impact. Despite the fact that DDRIO power shares a voltage regulator with memory, the power is typically assigned to the CPU for Running Average Power Limit (RAPL) usage models. This is done in order to effectively manage thermals within a power budget. It also allows the CPU to trade off unused power (and thermal) headroom back to the CPU cores when underutilized. This can be useful, since many high DDRIO power workloads do not require heavy core power, whereas core-centric workloads tend to have low to moderate DDRIO usage.

[6]Platforms that leverage buffered memory solutions (such as Haswell EX) have more complicated power delivery designs, and may run the DDRIO and DIMMs on separate voltage regulators.

Workload Behavior

Workloads tend to either demand very high memory bandwidth (≥ 80% peak) or be relatively insensitive to memory throughput (< 30% peak). There are always exceptions to the rule, but this is a trend that can be observed across a range of server workloads. Many of the workloads that fit into the high bandwidth category come from the high-performance computing segment, or could benefit from data structure optimization to improve cache locality. Memory power has a moderate dynamic range even without memory power management features. This is particularly the case with one-DPC configurations. It tends to be less apparent with two-DPC and three-DPC configurations, since on average, the percentage of traffic that goes to a given DIMM is cut (in half or in a third), reducing the actual dynamic range of the DIMM bandwidth. As an example, with a one-DPC DR configuration, scaling bandwidth from 20%–80% increases memory power by ~1.5 times.

Memory Reliability Features

A number of reliability features exist for memory that can have an impact on the power drawn by a given workload.

Memory Lockstep is a reliability feature where a single 64 B piece of data is stored across two DIMMs on two different memory channels. Since DDR3 and DDR4 work in BL8 mode, a single read or write actually fetches 128 B of data from the memory, increasing the amount of memory bandwidth that most workloads will consume. Lockstep tends to only be used in environments where high reliability is required because it both increases memory power and tends to have a measurable performance impact.

Patrol Scrub is a memory reliability feature that is typically enabled on all server CPUs by default. This feature attempts to walk through all of the memory space more or less every 24 hours, reading each line and checking the ECC. The goal is to identify errors while they can still be corrected. A single channel on each socket is typically scrubbed at a time. In certain situations this can result in channels not entering self-refresh because this blocks scrubbing, thus increasing memory power of idle systems. Patrol scrub is generally a low-cost method for reducing exposure to uncorrectable errors, and the added power cost is generally worth that reduction in exposure.

CPU I/Os

In addition to memory, there are a number of additional I/O capabilities that exist on modern server processors including interconnects that connect multiple sockets together (such as Intel QPI) as well as PCIe, which provides connectivity to devices like network cards and storage.

CPU Interconnect

In multi-socket systems interconnects exist that connect the different sockets to each other. These interconnects are used to maintain coherency across the sockets, to provide a communication channel between the sockets, and to connect memory that is connected from one socket to the other. In order to provide high performance and

prevent the coherency overhead from slowing down the performance of the cores, these interconnects are required to be high bandwidth and low latency and consume a non-trivial amount of power.

On the Sandy Bridge EP processors, the QPI interconnect consumes ~5 W of power per socket. Much of this power is consumed in the I/Os, and therefore it does not scale down significantly from one process generation to the next.

Because of their moderate power draw, these interconnects are most efficient on higher power processors that can amortize the cost of the power. Using a 5 W multi-socket coherent interconnect to hook up two 20 W processors is typically not worth the overhead. Rather than spending power on the I/Os, you are better off simply using a higher power single socket processor.

Link Power States

Power management of an interconnect is no different from anything else at a high level. However, one typical constraint is that it is difficult to scale the voltage of an interconnect in order to efficiently scale the frequency. As a result, reducing frequency can save power, but this is not always the most efficient decision. Table 3-6 illustrates some of the power states available on Intel's QPI 1.0.

Table 3-6. *QPI 1.0 Link Power States*

State	Name	Power	Granularity	Description
L0	Link Active	100%	–	Link active and running at full size and frequency. Provides maximum bandwidth at minimum latency.
L0s	Link Sleeping	~50%	Per direction	Subset of lanes asleep and not actively transmitting data. Not possible to send any information. Some lanes (clocks, etc.) still active, allowing for fast wakeup.
L0p	Partial Link Active	~75%	Per direction	Similar to L0s state, but a subset of the data lanes remain awake (typically half, but anything is possible). Bandwidth is reduced and latency for transmitting data increases.
L1	Link Down	< 10%	Entire link (both directions)	Link is completely powered down. In order to transmit data, it must be retrained.

L0p is a very useful power management state, particularly at low system utilizations. Many server workloads, particularly in the enterprise area, do not make heavy use of either memory bandwidth or the interconnect bandwidth. As a result, the loss in bandwidth from cutting the link in half has minimal impact on the actual performance or throughput of the system. L0p does, however, increase the amount of latency that it takes to transmit a data packet from one socket to the other. Data packets typically contain 64 B of data, and the link itself is much smaller than this. This can add ~10 ns of latency for such transfers. Although this latency is mostly inconsequential at low system utilizations, it can cost 1%–2% peak performance on workloads that are very latency sensitive and transmit significant data from one socket to the other.

L1 is an excellent state for idle systems. Although it takes multiple microseconds to wake a link back up, this is commonly "free" if there are long-latency actions being performed in the system (memory self-refresh, ramping voltage from a retention level to an active level, etc.). As a result, L1 is typically used during package C-states. Using it more aggressively during active states tends to result in performance glass jaws and even platform power increases.

L0s was a state that was productized on early QPI generations, but it has since been not supported. L0s is theoretically most useful if workloads exhibit bursty behavior between being active and completely idle. With a coherent interconnect and server workloads, it is uncommon to find periods of *no* traffic that last longer than a few hundred nanoseconds unless the system is completely idle. A trickle of coherency and communication traffic between sockets always seems to exist, particularly in real workloads that do not exhibit perfect NUMA locality. In situations where the system is indeed idle, L1 provides the necessary idle power savings.

■ **Note** L0s support has been removed from recent CPUs due to minimal power savings upside. L1 is only used during package C-states, where its latency can be hidden by other components during a wakeup.

Dynamic control of QPI frequency is not performed today. By reducing the frequency of the interconnect, not only is the bandwidth reduced, but the latency for transmitting data packets increases. This is particularly the case with L0p. This impacts the performance of the cores, which can spend more time stalled waiting for data to be returned to them. Not only does this impact the peak performance of the system, but it can even reduce the power/performance efficiency across a range of utilizations.

PCIe

The PCIe specifications provide standardized mechanisms for saving power. These link states are used across the wide range of devices that make use of PCIe (from low-power devices to servers). When it comes to PCIe link power management, server CPUs today are typically slaves to the devices that are connected to them. The devices themselves (through their own dedicated driver/firmware/hardware) initiate the transitions into power management states. The CPU is able to send negative-acknowledgment (NACK) requests but never initiates them.

Link Power States

PCIe uses L0s and L1 states. Unlike QPI, there is no defined partial width (L0p) state. However, it is possible to dynamically change (or statically configure) the link width that a device uses through the *upconfigure* flow. Table 3-7 provides an overview of the power management states used by PCIe.

Table 3-7. *PCIe Link Power States*

State	Name	Savings	Exit Latency	Granularity	Description
L0	Link Active	–	–	–	Link is active and running at full size and frequency. Provides maximum bandwidth at minimum latency.
L0s	Link Sleeping	~20 mW per lane, per direction	Microseconds	Per direction	Subset of lanes asleep and not actively transmitting data. Not possible to send any information. Some lanes (clocks, for example) are still active, allowing for fast wakeup. L0s is initiated autonomously by the link layer (no OS/driver interactions).
L1	Link Down	~100 mW per lane	Microseconds	Entire link (both directions)	Link is powered down. In order to transmit data, it must be retrained. Can be used dynamically at runtime. L1 can be triggered both autonomously by the connected device (ASPM L1) or through an OS/Driver call (L1-soft).
L2	Link Off	~125 mW per lane	Milliseconds	Entire link (both directions)	Saves slightly more power than L1. Generally used for unconnected links and links that are disabled at boot. L2 is only initiated through a software request.

■ **Note** Power savings is design dependent. These numbers provide a reference point.

L1 can be used during idle periods but has typically not been heavily utilized because of the latency impact and minimal power savings at the wall relative to overall socket power. Drivers and devices are typically tuned to use this state in only the most-idle conditions to avoid wakeup cost. With deep package C-states, the wakeup cost of L1 can generally be hidden since the CPU is informed about the wake and can perform other wakeup actions in parallel. L1 is becoming more interesting in the microserver space where the I/Os contribute a much larger chunk of the node power.

■ **Note** L0s is not supported on recent server processors. It saves relatively small amounts of power with non-trivial exit latencies. It is generally better off leaving a port in L0 or allowing it to drop all the way to L1.

Link Frequency/Voltage

PCIe has gone through three generations. Each generation has a single specified frequency at which the device runs (see Table 3-8). Multiple voltage/frequency points are not supported. However, the newer generation devices support (by rule) backward compatibility to the prior generation's frequencies. Changing frequency requires a full link retrain and is typically not performed dynamically at runtime today (although it is supported). Voltage is constant across generations/frequencies. When a PCIe 2.0 device is connected into a processor that supports PCIe 3.0, the device can only operate in PCIe 1.0 or 2.0 modes.

Table 3-8. *PCIe Generations*

Generation	Frequency	Theoretical ×8 Bandwidth
PCIe 1.0	2.5 GHz	2 GB/s
PCIe 2.0	5 GHz	4 GB/s
PCIe 3.0	8 GHz	7.88 GB/s

Although PCIe 3.0 only increased frequency by 60%, it was able to almost double the peak throughput. This was due to more efficient encoding and better use of the available wires for actual data.

Link Width

The number of lanes of PCIe impact the bandwidth that can be pushed through a link. Double the lanes, and the theoretical peak bandwidth is also doubled. The maximum number of lanes that a device is able to use is a function of both the device itself as well as the slot that it is connected to. Some systems may also allow the width of certain slots to be configured by BIOS. This can save a small amount of power but is generally not significant and can significantly reduce throughput. This could be useful for benchmarking but is not something that would generally be recommended for production systems.

PCIe devices can dynamically change the number of lanes in use at a given point in time. This can be done through software drivers (through the *upconfigure* and *downconfigure* flows) or through autonomous linkwidth change, allowing them to save power when lower bandwidth is required. These flows can be thought of as a way to reconfigure the link at runtime by restarting it with a different width (the lanes that are no longer used are no longer driven, reducing power). Reconfiguration typically takes microseconds, during which time the link is unavailable to transmit data. This flow has not been aggressively productized in server PCIe devices to date. These modes typically save relatively small amounts of overall system power and can cause non-trivial impacts to performance and latency in server usage models.

Hot Add

Many servers support Hot-Add flows. These allow PCIe cards to be inserted into the system at runtime without a reboot. However, this comes at a cost. In order to support Hot-Add, the lanes periodically cycle through a *DETECT* state that consumes moderate power. For a ×8 lane, this can add on the order of 100 mW of power on average. Through BIOS, PCIe lanes can be forced into an L2 state so that they do not perform this detection.

D-states

Similar to core C-states, PCIe devices can use D-states that indicate that a device is powered down. D-states are commonly used in phones, tablets, and laptops under idle conditions. They are not common under active load in servers. D-states are traditionally handled outside the CPU by PCIe devices that are connected to the platform with no interaction with the CPU (except side effects like the L1 state being used). As traditionally discrete devices are integrated into SoCs, this may change.

Summary

Memory can consume a large percentage of the overall power “pie” in many server systems. This is particularly the case in deployments that depend on large memory capacities. CKE and self-refresh can save significant amounts of power with almost no impact to the performance of the system. Making use of these capabilities is critical for achieving power efficiency in deployments with large memory capacities.

Selecting the correct type and configuration of memory can also have a large impact on energy consumption as well as performance. Building systems with at least two ranks per channel tends to result in the best performance across a wide range of workloads. Larger capacity devices tend to provide additional capacity at lower power cost than more DIMMs or more ranks, but they can also be cost prohibitive.

I/O power has historically been a much smaller contributor to overall system power, and getting overly aggressive with power optimizations in this area can be counterproductive. This is particularly the case with high-power CPUs. I/O power does become much more significant on CPUs with low power draw such as microservers and embedded devices.